Highly scalable parallel simulator of a host-parasite system

Michel Langlais, Guillaume Latu, Jean Roman, Patrick Silan

Highly scalable parallel simulator for a host-parasite system*

Michel Langlais[†‡] Guillaume Latu[†§] Jean Roman[†¶]
Patrick Silan[‖]

keywords numerical simulation, high performance computing, parallel algorithmic, host-parasite system, fish.

proposed running title HIGHLY SCALABLE PARALLEL SIMULATOR

1 Introduction

Life science models attempt to explicitly handle ecological processes that have significant effects, for example on population dynamics. Analytical models have not yet been able to describe and explain the complexity of all population dynamics issues. Computer simulation has proven to be a developmental tool complementary to theoretical and experimental modelling. The main features of a simulation involve: developing a theoretical model, generating a solution using an appropriate numerical algorithm, and subsequently analyzing the results. By building up a representation of the phenomena under consideration, and testing it under controlled conditions, scientists can gain a deeper understanding of the system. This leads to increasing the pool of knowledge, altering preconceived assumptions, or predicting the evolution of a biological system. Computational power is necessary to deal with many types of these models [14, 15]. Certain classes of complex phenomena, or space-related modelling approaches, or simply

*This work is supported by an ACI bio-informatique and integrated within the ScAlApplix project of INRIA UR Futurs.

†ScAlApplix project, INRIA UR Futurs, http://www.labri.fr/scalapplix

‡UMR CNRS 5466, Université Victor Segalen Bordeaux 2, 146 Léo Saignat, 33076 Bordeaux Cedex, France, Michel.Langlais@sm.u-bordeaux2.fr

§LSIIT, UMR CNRS 7005, Université Strasbourg 1, Boulevard Sébastien Brant, 67412 Illkirch Cedex, France, latu@icps.u-strasbg.fr

¶LaBRI, UMR CNRS 5800, Université Bordeaux I & ENSEIRB, 351 cours de la Libération, F-33405 Talence, France, roman@labri.fr

the level of detail of a model induce frequently large amount of computation and memory consumption. Likewise a multi-scale approach adressing multiple phenomenon could imply intensive computations. We shall describe such a model related to population dynamics, wherein ecological processes such as environmental variations (macro to micro scale), population interactions (meso scale), and individual biology (micro scale) are deeply interlaced.

Let us notice that parallelism is intensively used in the active field of discrete-event simulation programs [7, 22]. General-purpose environments are built on this concept of parallel discrete-event simulation. In the last decade the literature described several simulation languages and simulation environments that allow the construction of more specific types of ecological simulations. Some are centered on spatially explicit models [3, 6, 17], others on individual-based models [20], or even on cellular automata [21]. All these quoted environments use distributed algorithms and work on parallel machine or inter-connected workstations. They provide a more-or-less easy access to user interface, simulation management, and analysis tools. But specific models may not fit into these frameworks, and one may have to design a domain-specific application for solving a particular problem. In such cases, one needs an efficient implementation both to have a usable tool and to make it possible to validate the model (see for example [1, 16]). Parallel simulation has already proven to be an invaluable tool for population dynamics through simulation environments and specific applications.

This article deals with a performant parallel solution for a host-macroparasite system model in a marine environment. In a previous article [11], Langlais and Silan described the model we use which is both deterministic and stochastic. In this model, detailed phenomena are reproduced, such as spatial and temporal heterogeneities. Many factors are taken into account to simulate the model very precisely and get a reasonable level of realism. It constitutes a spatial model usually characterized by the discretisation of the problem domain into finer elements. But, unusually, the spatial decomposition consists in a distribution of age structured parasites on hosts instead of geographic distribution of entities in a 2D/3D environment; we will explain this point later. One novelty of our approch lies in accurately modelling processes belonging to individual and population scales. Our first goal in the simulation of such a model is to validate it and to discover the hierarchy of various mechanisms involved in the host-parasite system. The second one is to understand the sensitivity and the stability of the model with respect to initial conditions. The third one is to provide results that can be applied to devise prophylactic methods for fish farming, and more generally in epidemiology.

A sequential simulator has already been developed [4, 5], but computation time for one simulation took longer than one month. It is very important to reduce this duration to have a usable tool. Existing simulation environments are not adapted to the new approach used in our model, which includes age structure for parasites, the distribution of parasites on the host population, and the aggregation of parasites on hosts, when it appears. Because one single simulation may cost up to 105 TeraFLOP or even 1.45 PetaFLOP, parallelism

is required to experiment and explore this model. High-performance computers offer an opportunity to study such a realistic system.

In this work, we present the main issues concerning a parallel version that improves the previous simulator in terms of execution time. We present the complexity of computations and the sequential optimization. Then we develop a new parallel algorithmic solution and show a cost analysis. We investigate its performance and high scalability on IBM SP2, IBM SP3, IBM Regatta and SGI Origin 3800. Finally we give a brief validation of the model derived from real simulations. As a contribution to computational science, the work presented here is at the interface of ecology, mathematics, and computer science. This project is a collaborative effort in an interdisciplinary approach: population dynamics with C.N.R.S, applied mathematics and computer science with University of Bordeaux 1, University of Bordeaux 2, INRIA and University of Strasbourg 1.

2 Bio-mathematical model

Overview of the model

A formulation of the evolution of a discrete model is given by :

$$E_S(t + \Delta t) = f(E_S(t), E_P(t + \Delta t))$$

where $E_S(t)$ is the set of state variables at time t and $E_P(t)$ is the set of parameters. The state variables get updated every iteration whereas parameters are either constant or adjustable by the user. A numerical simulation is mainly intended to describe the evolution of two populations, fish (*Dicentrarchus labrax*) and parasite (*Diplectanum aequans*), over one year with a time step of $\Delta t =$ 2 days. Environmental conditions, such as water temperature, and biological data, are used in the simulations; they are included into $E_P(t)$ just as the parameters of the host-parasite model. $E_S(t)$ includes for example the number of eggs, of juvenile parasites, of adult parasites, of hosts, the ratio of over-parasitized hosts, *etc.* The final goal is to obtain values of the different variables in $E_S(t)$ at each step of the simulation. The global algorithm is sketched in Fig. 1.

Description of the model

We consider that only a surfeit of parasites can lead to the death of a host. A detailed age structure of the parasites on a given host is required because only adult parasites lay eggs, while both juvenile and adult parasites have a negative impact on the survival rate of hosts. The population of parasites is divided into $K = 10$ age classes, with 9 classes of juvenile parasites and one large class for adult parasites. To deal with heterogeneities in both populations, the model has a three dimensional data structure (ignoring the time dimension). Let N be that structure, and let $N_k(i, l, t)$ be the distribution of hosts having l parasites, i being older than $k\Delta t$. Thus, there is an age structure for parasites (variable

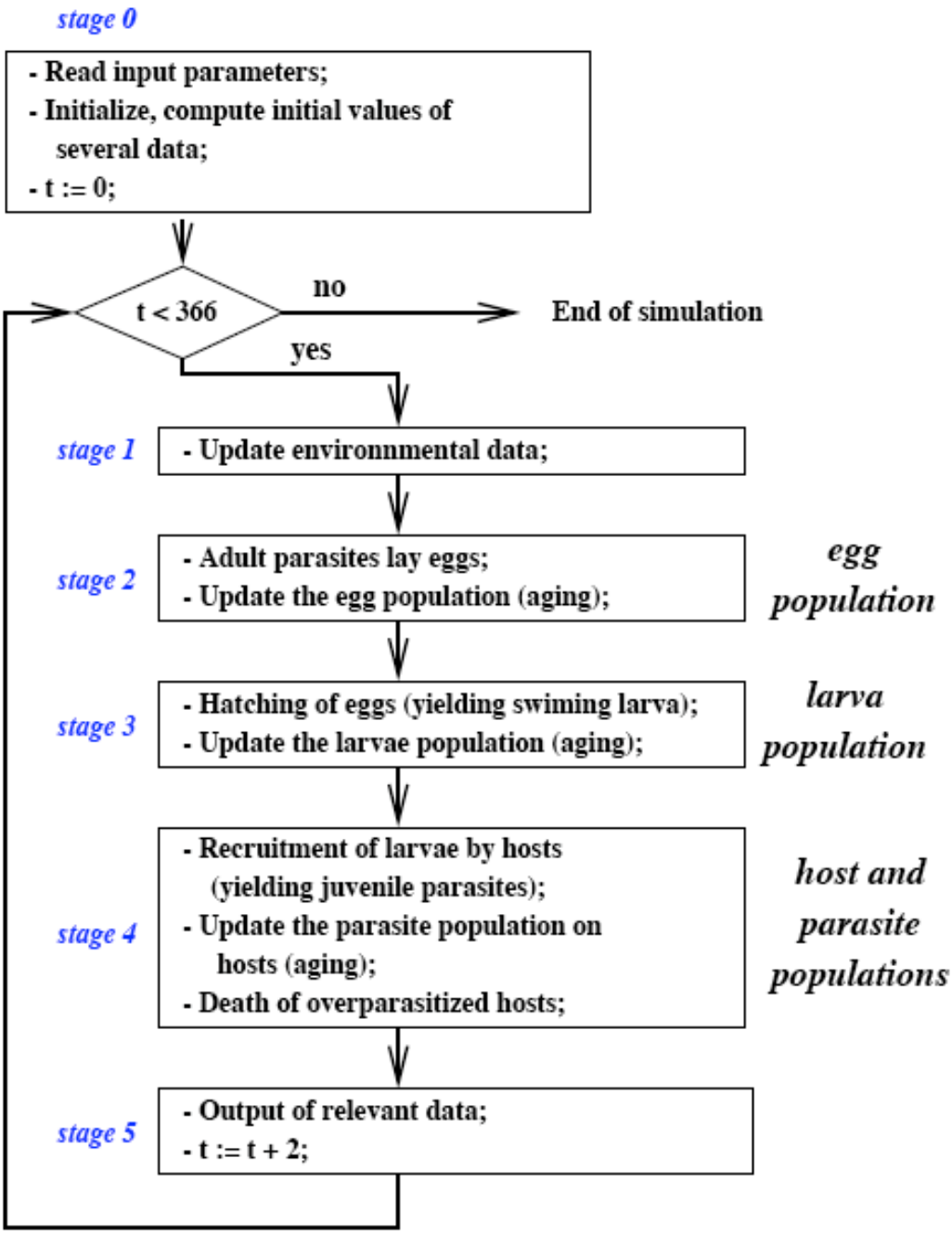

Figure 1: Global algorithm

k); the fish population is structured into groups of hosts having the same total number of parasites (variable l); the parasite distribution on a fish is given by the variable i. The complexity of the dataset comes from the detailed level of modelling we want to reach. The update of N belonging to $E_S(t)$ is very costly; it corresponds to the stage 4 of our algorithm (Fig. 1). The elementary events in this stage are quantified into probabilistic functions p, φ, δ which drive the update of N. See table 1 for explanations of the meaning of these functions.

In this model, environmental parameters are monthly-averaged and influence the biology of parasites. Every two days (the time step), adult parasites reproduce on a fish (infrapopulation level), larvae settle on fishes (individual fish scale) possibly. Each parasite age class has a specific mortality rate (cohort scale). There is an interdisciplinary challenge in this multi-scale model from an ecological point of view. From a mathematical point of view, we must take care of the consistency of computation because several variables are representing the same reality at different levels as, for example, $R(l,t)$ and $N_k(i,l,t)$. But this model is multi-level too in terms of the ecological structures we are interested in. The amount of information at each level is heterogeneous. Furthermore, there are highly nonlinear interactions bewteen the different model components. There is a challenge in specifying and validating a model with such multiscale processes.

Update of $N_k(i,l,t)$

The data stored in $N_k(i,l,t)$ is then modified at each time step (see Eq. 1), according to the following arguments. Let us consider a host having l parasites among which i are older than $k\,\Delta t$, $k \geq 2$ (see Fig. 2). At time t, this host had at least i parasites older than $(k-1)\,\Delta t$, say d, among which $d-i$ died from natural death between times t and $t+\Delta t$; furthermore the total load of parasites at time t was c among which $c-m$ died from natural death before time $t+\Delta t$, $0 \leq d-i \leq c-m$, yielding a recruitment of $l-m$ larvae. Next, the host survived a load of c parasites.

Adding up one has

$$\forall l \in [0, S(t)], \forall i \in [0,l], \forall k \in [2,K] \ ,$$

$$N_k(i,l,t+\Delta t) \ = \sum_{m=i}^{l} \sum_{c=m}^{S(t)} \sum_{d=i}^{c-m+i} N_{k-1}(d,c,t)\ \Pi(l,i,m,c,d) \ , \qquad (1)$$

wherein

$$\Pi(l,i,m,c,d) = p(c)\,\varphi(l-m,c)\,\delta(d-i,d)\,\delta(c-m-d+i,c-d) \ .$$

$S(t)$ is limited to the minimum number of parasites which is considered lethal for a fish (the biological hypothesis corresponds to 800 parasites for hosts younger than one year). The accumulated time of this update during the simulation can take up to 99% of the global simulation time. Our effort of speeding up the application has been mostly concentrated here. Before studying this computation, we present two other update formulae.

Symbol	Meaning
$p(l)$	probability for a host to survive a burden of l parasites over Δt;
$\varphi(j,l,t)$	probability that j larvae are recruited by a host having l parasites at time t;
$\delta(u,l)$	probability that u parasites die among l parasites older than $k\,\Delta t$;
$N_k(i,l,t)$	density of hosts having l parasites, i being older than $k\,\Delta t$;
$R(l,t)$	density of hosts having l parasites at time t;
Δt	time step of 2 days;
K	number of age classes: 10;
l_{lethal}	threshold above which a host cannot survive parasitism, *i.e.* $l_{lethal} = 800$ for a host having less than 1-year old;
$S_{max}(t)$	is an estimation of the maximal number of parasites fixed on a host;
$S(t)$	$= min(S_{max}(t), l_{lethal})$.

Table 1: Symbol table

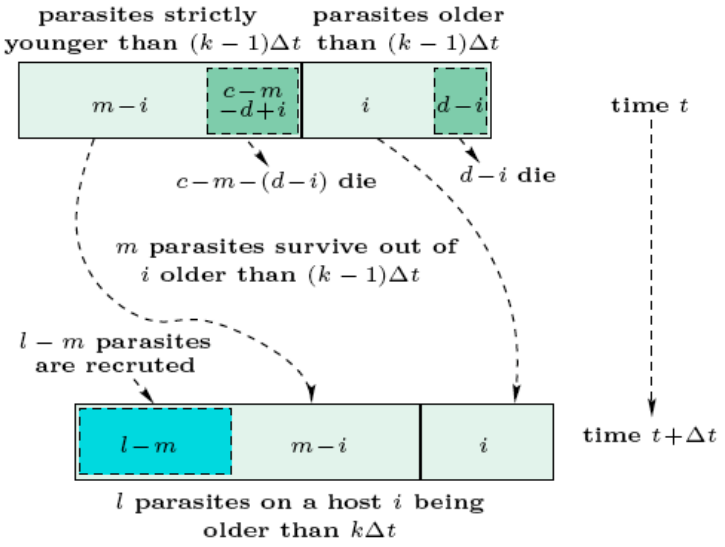

Figure 2: Evolution of the parasite population on a host having c parasites at time t

Update of $N_1(i,l,t)$ and $R(l,t)$

Similar arguments to those used for (1) hold for the update of $R(l,t)$ (density of hosts having l parasites) and $N_1(i,l,t)$. Let u be the number of dead parasites and j be the number of recruited parasites at time t, we have

$$\forall l \in [0, S(t)], \forall j \in [0, l],$$

$$N_1(j,l,t+\Delta t) \;=\; \sum_{u=0}^{S(t)-j} R(j+u,t)\,\varphi(l-j,j+u,t)\,\delta(u,u+j)\,p(j+u)\;, \quad (2)$$

and

$$R(l,t+\Delta t) = \sum_{j=0}^{l} \sum_{u=0}^{S(t)-l+j} R(l+u-j,t)\,\varphi(j,l+u-j,t)\delta(u,l+u-j)\,p(l+u-j)\;. \quad (3)$$

In the following section, we will present the complexity evaluation of the main computation of the simulator, that is the update of $N_k(i,l,t)$. Then the design of this costly computation will be re-defined, vectorized and last, parallelized.

3 Parallel computation

3.1 Algorithm description

Introduction to the parallel algorithm

First we reorganized the computations of equation (1): some precomputations in temporary arrays allow direct memory access to the values of our probabilistic functions, while basic factorizations and inversions of sums have been done to reduce computation cost. Deriving (1) with a symbolic calculator leads to a polynomial complexity $W(S(t))$ with a highest degree term given by $\frac{27}{40}\,S(t)^5$ [13, see chapter 6]. This means that the cost of one update of N can reach $W(800) = 225$ TFLOP. This cost is very high, especially since a complete simulation needs 183 updates. In order to improve performance, our approach has been to *vectorize* the update of N in order to use BLAS subroutines [2]. A matrix formulation of the algorithm allows us to intensively use BLAS 3 and leads to large speedups.

Moreover, it allows us to avoid some computation redundancies that appear in partial sums of the update [12]. Then, we gain one order of complexity; $W(S(t))$ has a higher degree term of $\frac{55}{24}\,S(t)^4$ and $W(800) = 950$ GFLOP.

It leads to an algorithm where intermediate matrices A_1, A_2 must be filled from precomputed arrays. Each entry of the array A_1 is computed by multiplying two elements from a precomputed array where $\delta(.,.)$ is stored; this means a cost of $(c-i+1)\,(c-i+2)/2$ multiplications to fill A_1. The matrix A_2 is filled

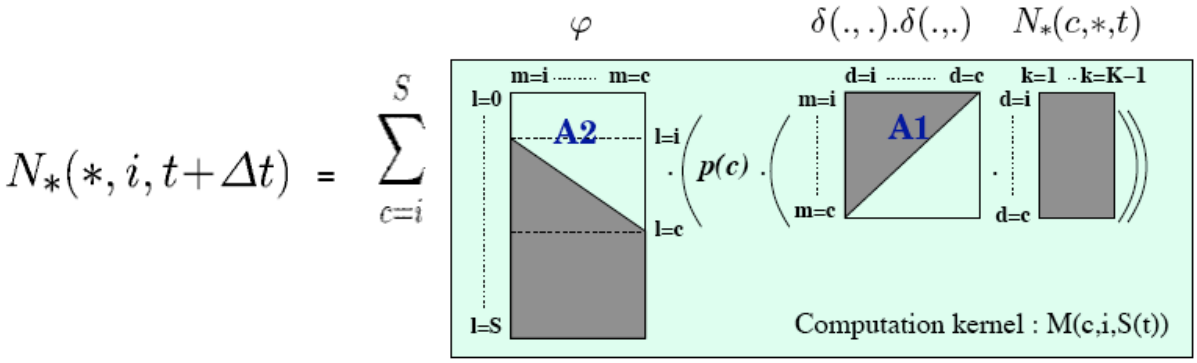

Figure 3: Matrix form of one update of N.

```
Procedure N_Update(S(t)) {
   For i:=0 to S(t) do {
   .   For c:=i to S(t) do {
   .   .    /* task M(c,i,S(t)) */
   .   .   initialization of A1, A2 matrices;
   .   .   Aux ← A2.(p(c).(A1.N*(c,*,t)));
   .   .
   .   .   N*(*,i,t+Δt) ← N*(*,i,t+Δt) + Aux;
   .   }
   }
}
```

Figure 4: Algorithm for the update of N

with a memory copy of an array associated with $\varphi(.,.,t)$. The update of the 3-dimensional data structure associted with N can be written[1] as illustrated in Fig. 3. Intermediate matrices contain by construction many zero elements which are represented by light grey sectors in Fig. 3. Since the initial formulation of the update of N, we have strongly reduced its cost and we can now study a parallel version.

In order to exploit BLAS performance, it is interesting to parallelize without cutting out the computation kernel $M(c,i,S(t))$ defined as the set of matrix multiplications in Fig. 3. Hence, this led to the algorithm described at Fig. 4 with two nested loops in variables i and c.

Before the study of the parallelization of this computation, one can look up for data dependencies to see what kind of parallelism is possible.

Data dependencies

In the following, we will refer to the data $N_*(c,*,t)$ as the "`row slab`" (c,t), and to $N_*(*,i,t)$ as the "`column slab`" (i,t) (see Fig 5). At iteration $(i=i_0, c=c_0)$ of the algorithm (Fig. 4), the row slab (c_0,t) is used to compute a contribution to the column slab $(i_0, t+\Delta t)$. The input data are row slabs and the results are column slabs. Two data structures for N are then required at each update, one for time t, the other one for time $t+\Delta t$, because the data at time t can not simply be overwritten. Furthermore, it is easy to see that if the column slab i_0 of time $t+\Delta t$ crushed the data N at time t, then this data N is corrupted and row slabs (c,t) (with $c \geq i_0$) can no more be used. Indeed, the column slab $(i_0, t+\Delta t)$ depends on all row slabs (c,t) for which $c \geq i_0$ (see Fig. 5).

Each structure N for time t and $t+\Delta t$ needs 24.5 MB (using double-precision reals); they will be dispatched over the processors of the parallel machine. Because row slabs are used and column slabs are generated, we must now study the distribution on processors of the slabs.

One dimensional and two dimensional distribution schemes

Let us suppose we parallelize only one loop of the algorithm; there are $S(t)+1$ tasks to map onto processors. As $S(t) \leq 800$, the algorithm has a coarse grain if we consider more than 100 processors. The parallelization of the outer loop i was evaluated in [9, 12] and gives good performances given in Table 3, p.34. This scheme is called 1D because indices i are distributed on a vector of processors.

But one can also consider the parallelization of the two loops. The elementary tasks $M(c,i,S(t))$ are then distributed over the parallel machine and the algorithm has a medium grain with $(S(t)+1)(S(t)+2)/2$ tasks to map onto P processors. This scheme is named 2D because the indices of each loop are distributed on a specific dimension of a grid of processors. As the grain is finer,

[1] The $*$ notation means to consider all elements of one given dimension.

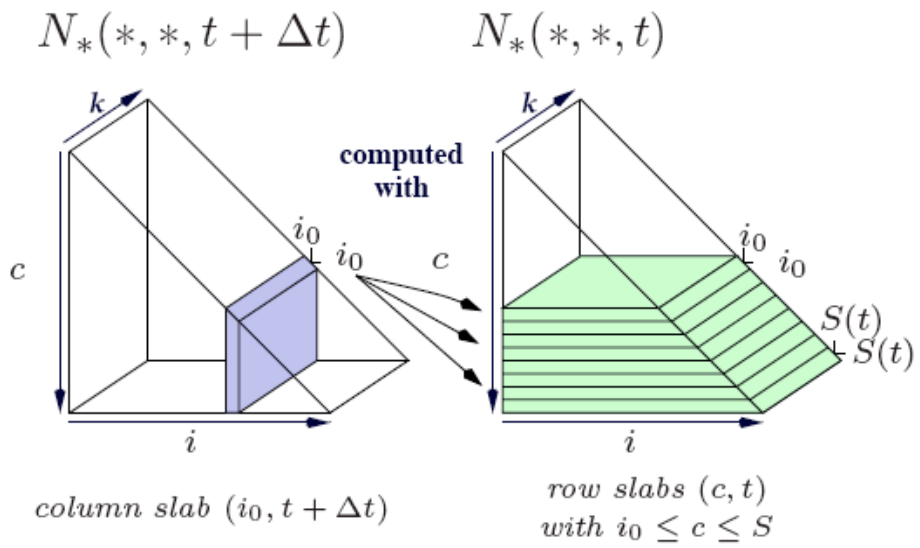

Figure 5: Dependencies of $N_*(*,*,t+\Delta t)$ on $N_*(*,*,t)$

	Number of processors: P				
	16	32	64	112	192
Run-time (sec.)	201.0	102.8	55.4	37.6	28.5
Relative efficiency	100.0 %	97.8 %	90.8 %	76.4 %	58.7 %

Table 3: Run time for an update of N with the 1D scheme on IBM SP2 ($S(t) = 700$ *i.e.* 560 GFLOP computation cost)

one can expect improving the scalability of the algorithm and getting a better relative efficiency than 59% for 192 processors (see Table 3). We will now give some explanation about computation and data distribution for this new 2D scheme.

Processor grid

Two loops are parallelized on P processors. We work on a grid of processors with $|G|$ lines in the first dimension and $|H|$ columns in the second, which is illustrated at Fig. 6; so we have $|H|.|G| = P$.

The indices i of the first loop of the algorithm are distributed on $|G|$ sets of processors. In each set, the indices c are mapped onto $|H|$ sets of processors. The actual distribution used will be discussed later on. If e is a column number of the grid of processor with $e \in [0, |H| - 1]$, let h_e be the group of processors responsible for a given set DH_e of indices c. If u is the line number of the grid, let g_u be the group of processors responsible for a given set DG_u of indices i. The processor belonging both to g_u and h_e is $p_{e,u}$. Thus, the processor $p_{e,u}$ has to run the tasks $M(c, i, S(t))$ with $i \in DG_u$ and $c \in DH_e$. The mapping function giving u or e in terms of i or c will be clarified in another paragraph. We introduce now the data distribution that will influence directly the choice of the mapping that we will effectively use. Let us observe that the 1D scheme corresponds exactly to the case $|H| = 1$ and $|G| = P$.

Data distribution

It is worth noticing in the algorithm of Fig. 4, that the tasks $M(*, i_0, S(t))$ compute contributions to column slab data $(i_0, t+\Delta t)$. In the group of processors g_u responsible for indices $i_0 \in DG_u$, it is interesting to map the column slab $(i_0, t+\Delta t)$ and the tasks $M(*, i_0, S(t))$ onto the same processor in order to execute matrix additions of the algorithm locally and then reduce communication costs. In each set g_u, all the processors will compute locally a partial sum for the column slab $(i_0, t+\Delta t)$, for $i_0 \in DG_u$. In a final step, these partial sums will be summed by the group g_u and store into the final column slab $(i_0, t+\Delta t)$. For each i_0 of DG_u, one processor of g_u is chosen to store the column slab $(i_0, t+\Delta t)$ at the end. This elected processor is given by a function named `mapi`(i_0). In the same way, one processor of each group of processors h_e is chosen to store the row slab (c_0, t) for $c_0 \in DH_e$; this processor is given by the mapping function `mapc`(c_0).

The update algorithm of N is rewritten in Fig. 7 and uses the mapping functions which have been introduced.

Because row slabs are used and column slabs are computed, a transposition step is needed to perform several successive updates. It appears in the parallel algorithm as the communication task 3. The cost in communication for this redistribution is equal to the storage amount of $N_*(*, *, t)$ which is $\frac{K\,S(t)^2}{2}$. It

	h_0	h_1	h_2	h_3	h_4	h_5
g_0	$p_{0,0}$	$p_{0,1}$	$p_{0,2}$	$p_{0,3}$	$p_{0,4}$	$p_{0,5}$
g_1	$p_{1,0}$	$p_{1,1}$	$p_{1,2}$	$p_{1,3}$	$p_{1,4}$	$p_{1,5}$
g_2	$p_{2,0}$	$p_{2,1}$	$p_{2,2}$	$p_{2,3}$	$p_{2,4}$	$p_{2,5}$

Figure 6: Processor grid: $p_{.,.}$ are processors; $g_.$ and $h_.$ refer to groups of processors

```
Subroutine N_UPDATE(S(t)) { /* on processor p_{u,e} */
   For all c ∈ DH_e do {
   .   /* communication task 1 */
   .   If mapc(c) = my_id then
   .        broadcast in group h_e of
   .        row slab N_*(c,*,t);
   .   Else
   .        receive N_*(c,*,t);
   .   For all i ∈ DG_u do {
   .   .   If (i ≤ c) then {
   .   .   .   /* computation task named M(i,c,S(t)) */
   .   .   .   initialization of A_1, A_2, A_3 matrices;
   .   .   .   Aux:=computation_kernel(i,c,N_*(c,*,t));
   .   .   .   N_*(*,i,t+Δt)=
   .   .   .      matrix_addition(N_*(*,i,t+Δt)+Aux);
   .   .   }
   .   }
   }

   /* communication task 2 : sum reduction */
   For all  i ∈ DG_u do {
   .    Sum of partial sums N_*(*,i,t+Δt) of
   .    all processors in g_u by the processor mapi(i);
   }

   /* communication task 3 : transposition */
   For c ∈ [0,S(t)] do {
   .   For i ∈ [0,c] do {
   .   .   If mapi(i) = my_id then send data N_*(c,i,t+Δt) to mapc(c);
   .   .   If mapc(c) = my_id then receive data N_*(c,i,t+Δt) from mapi(i);
   .   }
   }
}
```

Figure 7: Parallel algorithm for the update of N

is achieved using the MPI [19] subroutine `MPI_AllToAll`. In order to determine the mapping functions giving $p_{e,u}$ in terms of i or c, we have to evaluate the communication costs and to reduce, if necessary, the communication overhead with a specific mapping.

Communications

The communication cost is evaluated by considering each of the three communication tasks of the parallel algorithm one by one.

For the communication task 1, in each group of processors h_e, there is a local broadcast at each iteration $c \in DH_e$. The processor indexed by `mapc(c)` sends its row slab (c,t) corresponding to $K\,(c+1)$ real numbers to all processors in h_e. The data storage for N is $K\,S(t)^2/2$; in each set h_e, a ratio $1/|H|$ of this storage amount is sent or received locally during an update. At the end of one complete update, each processor has then handled approximately $K\,S(t)^2/(2\,|H|)$ reals among the row slabs. Because the processor is responsible on average for $1/|G|$ of the row slabs in its own group h_e, a processor broadcasts to the other processors of the group a ratio $1/|G|$ of $K\,S(t)^2/(2\,|H|)$, and receives from them $(1-1/|G|)$ of $K\,S(t)^2/(2|H|)$. Because $P = |H|.|G|$, the global communication cost on the parallel machine is then $(|G|-1)\,K\,S(t)^2/2$ (total of received communications). We observed that the complexity of the parallel algorithm is $55\,S(t)^4/(24\,P)$ plus the overhead due to parallelism. Whatever the overhead may be, the global cost of communication task 1, *i.e.* $(|G|-1)\,5\,S(t)^2$, is asymptotically less than the computation cost for $P \ll S(t)$. In practice the communication task 1 is pipelined and uses asynchronous non-blocking communication. That means that while computation is done, row slabs are sent in advance in order that computation overlaps communication. As the communication cost is less than computation cost when $P \ll S(t)$, we observe in practice a full computation/communication overlap.

For the communication task 2, in each set g_u summations of partial sums are processed to compute $N_*(*,i_0,t)$ with $i_0 \in DG_u$. At one iteration i_0 of the communication task 2, for one given column slab (i_0,t), all processors of the group minus one (the owner of column slab (i_0,t), that is `mapi(`i_0`)`) send their contributions to the processor `mapi(`i_0`)`. The data storage for N is $K\,S(t)^2/2$; in each set g_u, a ratio $1/|G|$ of this storage amount is handled during the communication task 2, that is, $K\,S(t)^2/(2\,|G|)$. There are $|H|$ processors in the set of processors g_u, so one processor is really responsible for $K\,S(t)^2/(2\,|G|)$ of data divided by H in average. During the sum reduction, one processor receives from all other processors of the set g_u their contributions for the column slabs they are responsible for, that means $|H|-1$ processors. Let us keep the received communications for the evaluation of the total communication cost per processor, which means a communication cost per processor equal to $(|H|-1)\,K\,S(t)^2/(2\,|G|\,|H|)$. The global cost on the parallel machine for the communication task 2 is then $(|H|-1)\,K\,S(t)^2/2$. Let us notice that this value is symmetric to the communication cost found for task 1 with $|H|$ and $|G|$ switched. As the total parallel

computation cost is more than $55\,S(t)^4/(24\,P)$, the communication cost of task 2 could be predicted as negligible. The communication of task 2 cannot be overlapped by computation, but the use of the MPI [18] subroutine `MPI_Reduce` is able to achieve relative good performances. Some time mesurements are given in section 4.

The communication task 3 corresponds to the transposition of the data structure N. This step is necessary because two data structures for N are required with column slabs stored on one hand, and row slabs stored on the other hand. The vector of K elements $N_\star(c_0, i_0, t+\Delta t)$ belonging to the processor `mapi(i0)` must be simply transferred to processor `mapc(c0)`. By moving all $N_\star(c_0, i_0, t+\Delta t)$ to their new places, the transposition is done; thus the communication cost is the same as the storage amount of $N_\star(*, *, t)$ that is $\frac{K\,S(t)^2}{2}$. That parallel global transposition is achieved in our implementation using the subroutine `MPI_AllToAll`. It is negligible compared to the cost $W(S(t))$, and the performances of the implementation confirms this assertion.

This organization of communication and the use of efficient MPI subroutines lead in our implementation to a very little communication overhead, as showed in section 4. After this study of communication costs, the distribution of computation must be evaluated. The only source of possible load imbalance is now work distribution.

Task cost

The complexity $CM(c, i, S(t))$ of the computation kernel $M(c, i, S(t))$ (see Fig. 3) is given by its number of multiplications and additions. Different ordering of the matrix multiplications (presented on Fig. 3) were envisaged. The chosen order $A_2.(p(c).(A_1.N_\star(c, *, t)))$ allows us to reduce asymptotically the computation cost. Let $d = c - i + 1$, then we have:

- the filling of the intermediate matrix $A1$ implies $d\,(d+1)/2$ multiplications;
- the computation of the matrix $B1 \leftarrow (A1\,N_\star(c, *, t))$ implies $\frac{(K-1)\,d\,(d+1)}{2}$ multiplications and additions;
- the matrix operation $B2 \leftarrow (p(c)\,B1)$ implies $(K-1)\,d$ multiplications;
- the matrix operation $B3 \leftarrow (A2\,B2)$ implies $(K-1)\,(\frac{d\,(d+1)}{2} + d\,(S(t)-c))$ multiplications and additions;
- the addition of $B3$ to $N_\star(*, i, t+\Delta t)$ implies $(K-1)\,(S-i+1)$ additions.

It follows that:

$$\begin{aligned} CM(c, i, S(t)) =& d[\frac{(d+1)}{2} + (K-1)(d+1) + (K-1) + (K-1)(d+1) + 2(K-1)(S(t)-c)] \\ &+ (K-1)(S(t)-i+1)\ , \end{aligned}$$

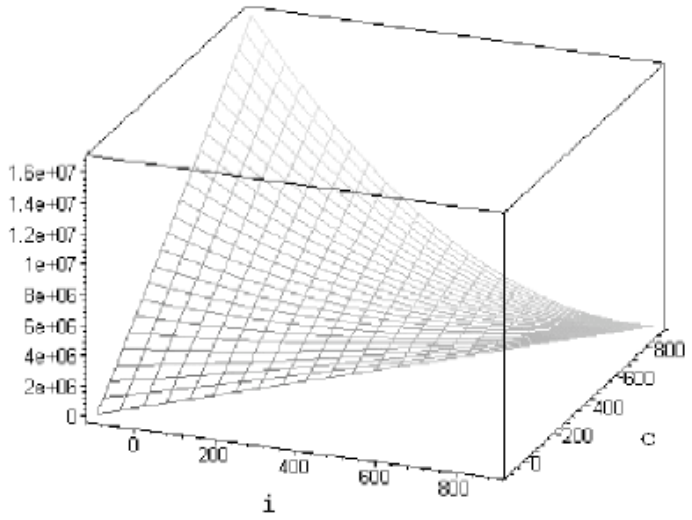

Figure 8: Curve of $CM(c, i, S(t))$ (for $S(t) = 800$) which gives the computation cost for each task $M(c, i, S(t))$

index e of group h_e	**0**	**1**	**2**	**3**
local indices c	21	20	19	18
	14	15	16	17
	13	12	11	10
	6	7	8	9
	5	4	3	2
			0	1

Figure 9: Snake distribution of indices c

index u of group g_u	**0**	**1**	**2**
local indices i	0	1	2
	5	4	3
	6	7	8
	11	10	9
	12	13	14
	17	16	15
	18	19	20
			21

Figure 10: Snake distribution of indices i

furthermore as $K = 10$

$$CM(c,i,S(t)) = \frac{1}{2}(c-i+1)\,[c-37\,i+36\,S(t)+92] + 9(S(t)-i+1). \qquad (4)$$

The sequential complexity W previously introduced can be directly inferred from the cost function CM:

$$W(S(t)) = \sum_{i=0}^{S(t)} \sum_{c=i}^{S(t)} CM(c,i,S(t)) = \frac{55}{24}\,S(t)^4 + \frac{293}{12}\,S(t)^3 + o(S(t)^3).$$

The task computation cost $CM(c,i,S(t))$ is displayed on Fig. 8. For $i \in [0,S(t)]$,$c \in [i,S(t)]$, the function CM is positive, decreasing in variable i and increasing in variable c. Furthermore, this function is convex in variable i (for c fixed) and convex in variable c (for i fixed):

$$\frac{\partial^2 CM}{\partial^2 c}(c,i,S(t)) = 1 > 0, \qquad \frac{\partial^2 CM}{\partial^2 i}(c,i,S(t)) = 37 > 0\ .$$

We deduce from that the following first property:

Property 1 *For $0 \le i \le c \le S(t)$, the function $CM(S(t)-c,i,S(t))$ is decreasing and convex for both variables i and c.*

Properties of the distributions

There are two main distributions to consider: the indices c are mapped onto the groups of processors h_e, and indices i are distributed on the sets g_u. As the communications take little time compared to the computation, our aim is to find a good computation distribution. An optimal distribution of the tasks $M(*,*,S(t))$ could be considered, but in order to study easily and precisely the parallel algorithm, we used a more regular distribution. In order to allow us a theoretical study we use the *snake distribution*.

The complexity of a task $M(c,i,S(t))$ is a decreasing function for the variable c. For the distribution of the index c, we use the biggest c of the update, which gives $c = S(t)$ as the first index to be mapped onto the processors. It means that we distribute the bigger tasks first and the smaller ones last to have a finer distribution. Because the complexity of a task $M(c,i,S(t))$ is an increasing function for the variable i, we take $i = 0$ as the first index to begin the distribution. Suppose that $|G| = 3$ and $|H| = 4$ for $S(t) = 21$, one gets the distributions of Fig. 9 and Fig. 10.

Hence, we achieve a well-balanced work load. We can now analyze the performance of this distribution of computations.

3.2 Cost analysis

Computation cost on processors

The distribution of tasks $M(c,i,S(t))$ is known and the complexity of each task is known too, thus the computation cost on each processor can be evaluated. The distribution is static, so a theoretical study of efficiency is even possible. In order to calculate this efficiency, one has to identify which processor has the most operations to execute, and to know the exact computation cost of this processor. We will assume that the grid of processors is a square of width Y. The processors are named $p_{e,u}$ with $(e,u) \in [0,Y-1]^2$. We have a finite number of elementary tasks $M(S(t)-c,i,S(t))$ with $0 \leq i \leq c \leq S(t)$ to distribute on the processors. The property 1 indicates that the number of operations of an elementary task $CM(S(t)-c,i,S(t))$ is a decreasing and convex function for variable i and variable c.

Proposition 1 *Suppose that $2Y$ divides $S(t)+1$. We distribute tasks with a bidimensionnal snake distribution on the grid of $P=Y^2$ processors, so we map the task $M(S(t)-c,i,S(t))$ on processor $p_{e,u}$ with the following rules (illustrated also by Fig. 11):*

- *let r_i be $i \bmod 2Y$,*

  if $0 \leq r_i \leq Y-1$ *then* $e = r_i$,
  else $e = 2Y-1-r_i$;

- *let r_c be $(S-c) \bmod 2Y$,*

  if $0 \leq r_c \leq Y-1$ *then* $u = r_c$,
  else $u = 2Y-1-r_c$;

Let $load(e,u)$ be the function giving the computation complexity on each processor $p_{e,u}$. We have the relation

$$\forall (e,u) \in [0,Y-1]^2 \quad load(e,u) \leq load(0,0). \tag{5}$$

Then, the processor $p_{0,0}$ has to perform more computation than the others.

This proposition is proved in [13, see Annex A]. It follows that the processor which has the greatest load is $p_{0,0}$ (if the grid of processors is a square of with Y, and if $2Y$ divides $S(t)+1$). Furthermore, we have observed that $p_{0,0}$ has the greatest number of operations to perform in many configurations without the hypothesis $2Y$ divides $S(t)+1$. As an illustration, an example of the number of operations to perform per processor is given on Fig. 12 for $S(t) = 800$ and $Y = 8$. Moreover, that observation is also true whenever we use a rectangular grid of processors.

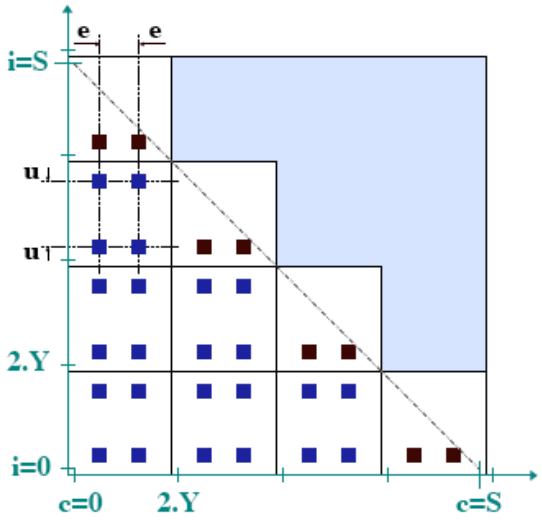

Figure 11: Black squares represent tasks $M(S(t) - c, i, S(t))$ mapped onto processor $p_{e,u}$

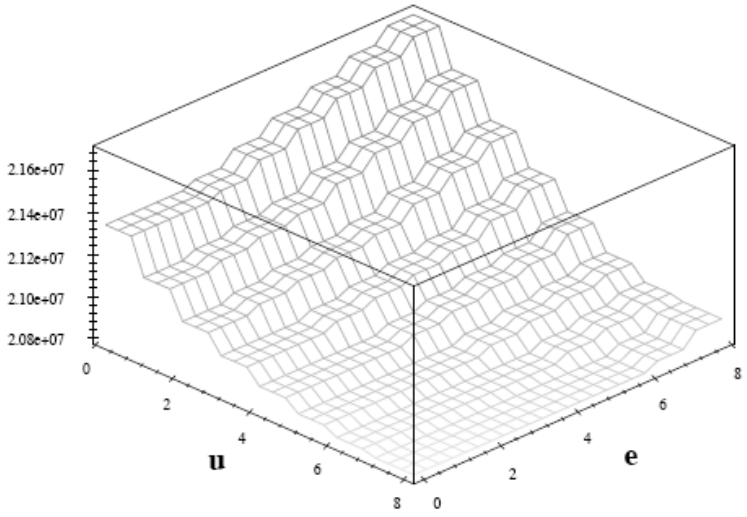

Figure 12: Number of operations $load(u, e)$ of all tasks M mapped onto processors $p_{e,u}$ $(S(t) = 800,\ P = 64,\ Y = 8)$

Overhead for the 2D scheme

The processor $p_{0,0}$ gives us the cost of the parallel algorithm, which is $Pload(0,0)$. As we have the sequential complexity, we will be able to determine the overhead and the efficiency of this algorithm. From this, one can derive a theoretical estimation, but one must consider that the complexity in terms of number of operations is different than complexity in time. In fact, the time of initialization of transient matrices is not negligible, there are caching effects because of matrices reusability in the kernel of computation, and last but not least the use of BLAS modifies the cost in time. One deduces that complexity in time differs from complexity in number of operations $W(S(t))$; but it is straight-forward to show that they have both the same asymptotical complexity $T_{seq}(S(t)) \approx \Theta(S(t)^4)$. In this section we will work only on the costs in number of operations which are theoretically known. Furthermore, we will suppose again that $2Y$ divides $S(t)+1$. Let us denote by Z the value $S(t)+1$. In such conditions the cost of task $M(c,i,Z-1)$ deduced from equation 4 is:

$$CM(c,i,Z-1) = \frac{1}{2}c^2 - 19\,c\,i + \frac{59}{2}\,c + \frac{37}{2}\,i^2 - \frac{95}{2}\,i + 29 + 18\,c\,Z - 18\,i\,Z + 18\,Z \quad .$$

Let $W_{//}(Z-1,Y)$ be the parallel cost on the $Y^2 = P$ processors. The parallel cost equals the number of operations on processor $p_{0,0}$ times the number of processors, that is

$$W_{//}(Z-1,Y) = Y^2\, load(0,0) \ . \tag{6}$$

The expression of $load(0,0)$ is a function of CM that depends on the snake distribution [13, see chapter 6]:

$$\begin{aligned} load(0,0) = & \qquad (7)\\ \sum_{j=1}^{q}\sum_{g=1}^{q-j+1} & [CM(Z-1-(2Y(g-1)), 2Y(j-1), Z-1) + \\ & \quad CM(Z-1-(2Yg-1), 2Y(j-1), Z-1) + \\ & \quad CM(Z-1-(2Y(g-1)), 2Yj-1, Z-1) + \\ & \quad CM(Z-1-(2Yg-1), 2Yj-1, Z-1)] \\ +\sum_{j=1}^{q+1} & [\ CM(Z-1-(2Y(q-j+1)), 2Y(j-1), Z-1) + \\ & \quad CM(Z-1-(2Y(q-j+2)-1), 2Y(j-1), Z-1) + \\ & \quad CM(Z-1-(2Y(q-j+1)), 2Yj-1, Z-1)] \ . \end{aligned}$$

The overhead of the algorithm $W_{ov}(Z-1,Y)$ is given by:

$$\begin{aligned} W_{ov}(Z-1,Y) &= W_{//}(Z-1,Y) - W(Z-1) \\ &= Y^2\, load(0,0) - W(Z-1) \ . \end{aligned}$$

At one time step t, let $X = Y - 1$ for the simplicity of following formal expressions. After a tedious derivation performed using a program for symbolic computation, we get:

$$W(Z-1) = \sum_{i=0}^{Z-1} \sum_{c=i}^{Z-1} CM(c,i,Z-1) \;=\; \frac{55}{24} Z^4 + \frac{61}{4} Z^3 + \frac{605}{24} Z^2 + \frac{49}{4} Z^2,$$

$$W_{ov}(Z-1, X+1) = \left[\frac{28\,X^2}{3} + \frac{137\,X}{12}\right] Z^2 + \left[17\,X^2 + \frac{117\,X}{4}\right] Z \;.$$

The domain of definition where the parallel simulator is planned to be used is nearly $Z \in [200, 2000]$, and $X \in [4, 40]$. For this use of the algorithm the overhead is very close to:

$$W_{ov}(Z-1, X+1) \;\approx\; \left[\frac{28}{3} X^2 + \frac{137}{12} X\right] Z^2 \;.$$

which leads to the following result:

$$W_{ov}(S(t), \sqrt{P}) \underset{S \to \infty}{=} \frac{1}{12} (112\,P - 87\sqrt{P} - 25)\, S(t)^2 \;.$$

We can now consider the effective differences between the number of operations to perform on a processor and the time of computation on this processor. It is reasonable to first suppose that the theoretical overhead is slightly different from the real one, but let say that the asymptotical behavior is the same. If we named the real overhead $T_{ov}(S(t), \sqrt{P})$, we have $T_{ov}(S(t), \sqrt{P}) = \Theta(P\,S(t)^2)$. For the 1D scheme [9, 12] the overhead was equal to $\Theta(P^2\,S(t)^2)$; then the benefit of the 2D scheme is actually significant.

Efficiency and scalability

Let $E(S(t), K, P)$ be the efficiency of the algorithm. It can be expressed as a function of the overhead, as stated in [8],

$$E(S(t), P) = \frac{1}{1 + \frac{W_{ov}(S(t), \sqrt{P})}{W_{seq}(S(t), K)}} \;.$$

Because of the expressions of $W_{ov}(S(t), \sqrt{P})$ and $W_{seq}(S(t), K)$, asymptotically we get:

$$E(S(t), P) = \frac{1}{1 + \frac{224\,P - 174\sqrt{P} - 50}{55\,S(t)^2} + o(\frac{1}{S(t)^3})} \;. \tag{8}$$

As $T_{seq}(S(t)) = \Theta(S(t)^4)$, and $T_{ov}(S(t), \sqrt{P}) = \Theta(P\,S(t)^2)$ we have $T_{ov}(S(t), \sqrt{P}) \ll T_{seq}(S(t))$ when P is negligible compared to $S(t)$. Under this last assumption, which corresponds to the way we use the simulator, we conclude that the algorithm is cost-optimal, that is $T_{//}(S(t), P) = \Theta(T_{seq}(S(t)))$. We can fix $E(S(t), P)$ and derive the formula of isoefficiency from (8) which drives to $S(t) = \Theta(\sqrt{P})$ implying a scalable algorithm.

4 Performances

4.1 Parallel computers

We performed our experiments on the IBM SP3 (and previously on the IBM SP2), the SGI Origin 3800 of CINES[2] and on the IBM Regatta of IDRIS[3]. The 28 nodes of the SP3 are 16-way NH2 nodes (375 MHz), with 16 GBytes of memory. A Colony switch manages the interconnection of nodes with a bi-directional theoretical bandwidth of 1GB/s. The code has been developed in FORTRAN 90 with the XL Fortan compiler and using the MPI message-passing library (IBM proprietary version).

The SGI Origin 3800 is a scalable shared memory multiprocessor system. In terms of parallel computer architecture, this machine is an cc-NUMA architecture (cache coherent Non Uniform Memory Access). With a frequency equal to 500 MHz, the theoretical peak performance is 1 GFLOPS per processor if the two independent floating-point units are busy. There is no fused multiply-add capability as for the IBM Power 3 and Power 4 processors. The level 1 instruction and data caches have a size of 32 KB, whereas the secondary unified instruction/data cache size is 8 MB. Two SGI machines of 512 and 256 processors have been used at CINES.

The last machine consists of eight IBM p690 Regatta nodes, each containing 32 Power 4 processors, for a total of 256 processors. The Power 4 processor has a 1.3 GHz clock rate, a 32 KB L1 data cache and a 64 KB L1 instruction cache. A L2 data cache of 1.5 MB is shared by an elementary block of two processors and a L3 of 512 MB is shared within a 32-way node. There are two floating point multiply-add units each of which can deliver one result per clock cycle, which gives a theoretical peak performance of 5.2 GFLOPS per processor. The Colony switch provides in this machine bi-directional network links of 200 MB/s each (two links per node).

4.2 Kernel optimization

The tasks $M(c, i, S(t))$ could be divided into four elementary subtasks (see Figure 3). First, A_1 is initialized and two matrices are multiplied: $aux_1 = A_1\, N_\star(c, *, t)$. Second, a scalar-matrix multiplication is computed: $aux_2 = p(c)\, aux_1$. After, A_2 is initialized and multiplied with another matrix: $aux_3 = A_2\, aux_2$. Finally, two matrices are summed up: $N_\star(*, i, t + \Delta t) = N_\star(*, i, t + \Delta t) + aux_3$. The costly part of this set of computations is the two subtasks involving an initialization followed by a matrix multiplication. This step must be very efficient because much of execution time is spent in this

[2] Centre Informatique National de l'Enseignement Supérieur (France).

[3] Institut du Développement et des Ressources en Informatique Scientifique (CNRS, France).

```
Subroutine algo_1(dm, dk, dn, B, C, g) {
   A : temporary matrix ;
   For k := 1 to dk do {
   .    For n := 1 to dn do {
   .    .      a_{k,n} :=  g(k, n);   /* initialization */
   .    }
   }
   C := B A;     /* multiplication use BLAS 3 subroutines (GEMM,TRMM) */
}
```

Figure 13: Algorithm `algo_1` solving the Ω problem

```
Subroutine algo_2(dm, dk, dn, B, C, g) {
 α : temporary matrix
 For k := 1 to dk by stride_k do {
 . For n := 1 to dn by stride_n do {
 . . For i := 0 to stride_k − 1 do {
 . . . For j := 0 to stride_n − 1 do {
 . . . .   α_{i,j} :=  g(k + i, n + j);
 . . . }
 . . }
 . . C(1 : dm, n : n+stride_n−1) := C(1 : dm, n : n+stride_n−1) +
 . .                B(1 : dm, k : k+stride_k−1) α(0 : stride_k−1, 0 : stride_n−1);
 . }
 }
}
```

Figure 14: Blocked algorithm `algo_2` solving the Ω problem

computation. It corresponds to the formal problem Ω defined by:

$$\underline{\text{Problem } \Omega}: \text{ Efficient computation of } \forall (m,n) \in [1,dm]\times[1,dn] \; c_{m,n} = \sum_{k=1}^{dk} g(k,n)\, b_{m,k}\,.$$

This formulation uses a matrix B of dimension $dm \times dk$, a function g defined on $[1,dk] \times [1,dn]$, and a $dm \times dn$ matrix C. In order to use the level 3 of BLAS, it is straight-forward to write the algorithm `algo_1` as presented in Figure 13. It solves the problem Ω using a combination of calls to GEMM and TRMM subroutines depending on the shape of matrix A.

The use of BLAS 3 primitives improve the kernel performances. However, one can observe that if the temporary matrix A leaves the data cache between the initialization and the multiplication, it leads to a non optimal situation (many cache misses). In such case, the matrix A could be loaded two times from memory to data cache, implying a larger memory bandwidth sollicitation. In order to solve this problem, we could mix the initialization of sub-matrices of A and their multiplication with B. If a sub-matrix (say α) of A stays in either one of the data caches from initialization to its multiplication, we therefore increase the temporal data locality. The blocked algorithm `algo_2` of Figure 14 has such a behavior. The blocking size along the dimension k and n are $stride_k$, $stride_n$ respectively. The algorithm is not blocked along the m dimension. To simplify, we suppose that $modulo(dn, stride_n) = 0$, $modulo(dk, stride_k) = 0$.

Let us notice that $dm = K-1 = 9$ is small in our application. The minimum complexity in number of memory references for the matrix multiplication of the Ω problem is $dk\,(9 + dn) + 9\,dn$, whereas the computational complexity is $18\,dk\,dn$. Asymptotically, the complexity in computation could represent eight-teen times the complexity in number of memory references. That case is not actually typical for a call to a GEMM subroutine. One could expect to have much more computations compared to memory references and to use actively the data cache to saturate the processor computational capability. Our matrix multiplication is not very far from a vector-matrix multiplication where the actual bottleneck is memory bandwidth when data are not cached. We have implemented a matrix-matrix multiplication adapted to our special case $dm = 9$ using classical techniques: loop unrolling, tiling for registers and caches, prefetching, array padding. Care has been taken to avoid memory copies because of the bandwidth bottleneck. To avoid the cost associated with function calls the code of matrix multiplication with $dm = 9$ were compiled within the `algo_2`. The blocking sizes of the algorithm pas_k, pas_n were determined by a sequence of benchmarks on the simulator for each considered machine. Results with the best sets of blocking sizes are presented in Table 2.

For each considered parallel architecture, the algorithm `algo_2` improves the performance compared to `algo_1`. Furthermore, on the Power 4 processor, the MFLOPS rates are almost doubled. For the SGI machine and the IBM SP3, near 60% of the peak performance is reached. This very good sustained performance is not achieved for the IBM Regatta because of a lower ratio memory bandwidth

processor	R14000, 500 Mhz		Power 3, WH2		Power 3, NH2		Power 4, 1.3 Ghz	
	time	MFLOPS % peak	time	MFLOPS % peak	time	MFLOPS % peak	time	MFLOPS % peak
Update of N, `algo_1`	111.8s	532 53 %	112.5s	528 35 %	86.8s	685 46 %	71.0s	837 16 %
Update of N, `algo_2`	97.1s	612 61 %	69.1s	860 57 %	66.4s	895 60 %	36.8s	1544 30 %

Table 2: Performances of both algorithms algo_1 and algo_2 used in the update of N on 16 processors for different superscalar architectures ($S(t)$=800, *i.e. 950 GFLOP computation cost for the update of N)*

	Number of processors: P					
	4	16	64	128	256	448
Execution-time (sec.) IBM Regatta	18 605	4 830	1 248	-	-	-
Execution-time (sec.) SGI O3800	44 336	11 175	2 896	1 488	784	518
Execution-time (sec.) IBM SP3 NH2	32 361	8 173	2 179	1 110	603	387

Table 4: Run time for a complete simulation (105 TFLOP)

over processor clock compared to the other machines.

4.3 Extensibility of the parallel kernel

We present now the performances for one update of N on a IBM SP2 (207 thin nodes 120 MHz, 256 MBytes of memory, interconnected by a TB3 switch). In a previous version of the parallel simulator, there was only a parallelization of the outer loop i (algorithm of Figure 4). The results of this simulator were presented in [9, 12] and are given in Table 3. We observe that from 16 to 112 processors, the relative efficiency is above 75%, whereas for much processors the relative efficiency decreases significantly.

From now on, we will show better results for the parallelization of the two loops i and c (2D scheme). The theoretical efficiency was established at formula 8. We have not been able to measure this asymptotic behavior experimentally because it would have required many more processors than we may have. The observed parallel efficiency is presented for different architectures at Figure 15. In fact, up to 448 processors on the SP3 and the Origin 3800, the efficiency for the computation kernel is nearly not degraded (above 94%). For all number of processors used (up to 448) and all parallel machines, the relative efficiency compared to 4 processors is larger than 90%. Then, the good performance of our parallel algorithm and the quality of the load balancing is observed in theory and in practice.

4.4 Scalability of the complete simulation

At Figures 16, 17 and 18 we observe the time proportion for different parts of the simulation program. When using several hundred of processors, the effect of Amdahl's law could be noticeable. In a parallel application, one must parallelize as many parts of the code as possible. The parallelization of several costly pieces of code in our simulator shows that time taken by computations other than for the update of N takes a relatively small percentage of the global time of one step. It is worth noticing that the parallelized "update of N_1" and "update of R" are scaling well even for different parallel machines. Nevertheless, the "transposition step" and the labeled part "other computation or idle time" appear to have a cost that increases with the number of processors. The reason is that collective communications involving all processors are required, such as MPI_AllToAll, MPI_AllGather or MPI_AllReduce. Consequently, the impact on scalability is linked to the performances of the network and of the MPI implementation on the target architecture. In our experiments, and for $S(t) = 800$, we have less than 1.3% of the total execution time spent in a sum reduction (it corresponds to communication task 2 on Fig. 7). The transposition step (communication task 3) correponds to less than 2.3% of the total execution time. These two communication tasks represent then a relative reduced cost compared to computation, even for a large number of processors.

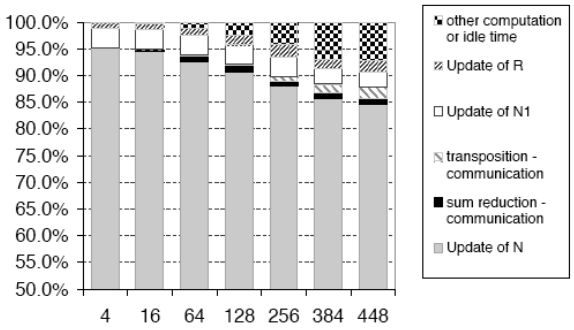

Figure 16: Time proportion of different parts of the program on a SGI Origin 3800 for an update of N ($S(t) = 800$)

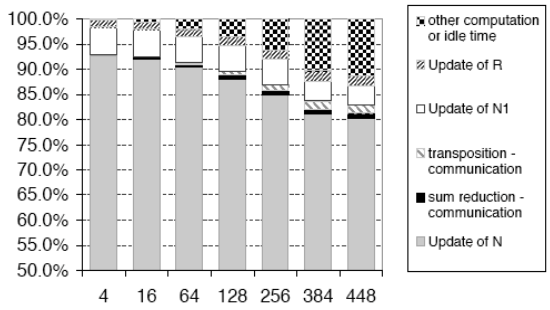

Figure 17: Time proportion of different parts of the program on a IBM SP3 NH2 for an update of N ($S(t) = 800$)

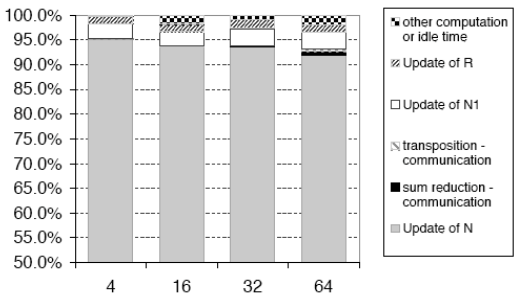

Figure 18: Time proportion of different parts of the program on a IBM Regatta for an update of N ($S(t) = 800$)

The load is well balanced, the communication cost is relatively small compared to computation, and the whole costly computation were parallelized. That explains why the parallel application is highly scalable. And if we have a look to Fig. 19, we see that the relative efficiency for one entire simulation is above 74 % for all the number of processors tested on the IBM SP3 and SGI Origin 3800. We notice that up to 64 processors the efficiencies are very close for every tested architectures. The algorithms used are very efficient and portable. The SGI Origin 3800 gets a better scalability than the IBM SP3 because of relatively higher communication performance. The table 4 illustrates the fact that above 128 processors, a costly simulation of 105 TFLOP lasts only less than 25 minutes. For 448 processors, this simulation lasts only 6 minutes and 27 secondes (IBM SP3). On a single processor of the IBM SP3 or the Origin 3800, it would have taken more than one day. Our goal of giving a usable tool is then achieved.

4.5 Numerical precision

When changing the number of processors, parallel computations are ordered slightly differently, which can cause small approximation errors. When using double-precision reals for computation, all variables observed during the entire simulation remain identical up to at least the first 9 significant digits, even if the number of processors is modified. But for simple-precision real computations, if the number of processors is changed, the results are the same for only the first 2 significant digits. Hence, double-precision improves, as usual, numerical stability but the cost in communication bandwidth (and in computation on old 32-bit architecture) is higher. However, we choose to use double precision computation for our simulation for the best quality of results achieved. Numerical stability is an important issue for biologists and mathematicians: the observed phenomena in numeric simulations must come from the underlying model and not from computation artifacts.

4.6 Petaflop simulation

In order to understand the sensitivity of the host-parasite system to the lethal threshold of parasite per host, we have to test some higher values for the l_{lethal} parameter. Depending on environmental conditions, the sea basses could be more or less heavily impacted by their burden of parasites. For the mathematical model, it means that $S(t)$ could be two times higher and reach 1600. At the computation level, the tasks $M(c, i, S(t))$ will consider larger matrices in average. That increases the difference of performance between `algo_1` and `algo_2`. For example, the execution times of the update N for $S(t)$=800 on the IBM SP3 NH2 was 23% lower with `algo_2` compared to `algo_1` (it can be deduced from Table 2). When we use $S(t)$=1600, the gain reaches then 43% (comparing the execution time of the update of N for both algorithms in the Table 5). So, this Table 5 shows the very good performances of the algorithm `algo_2`. A complete

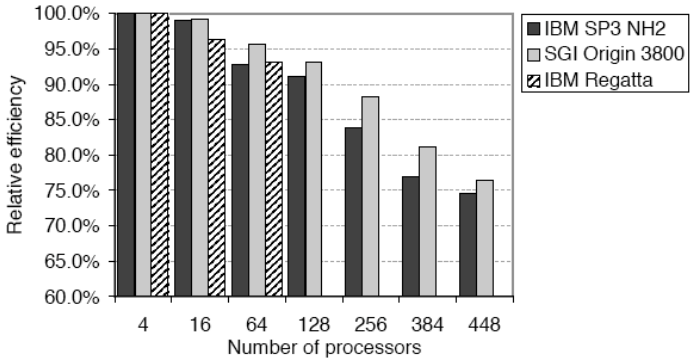

Figure 19: Relative efficiency for a complete parallel simulation ($S(t) = 800$, with a cost of 105 TFLOP

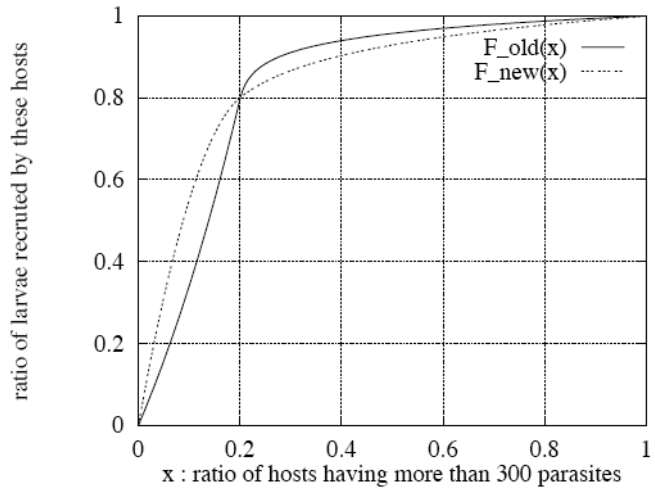

Figure 20: Two versions of the function giving the ratio of larvae recruited by hosts having more than 300 parasites

simulation of 1.45 PFLOP achieves to use above 50% of the peak performance of the parallel machine on 256 and 448 processors. The execution time of this simulation with `algo_2` on one processor would reach 3 weeks. On 448 processors, the update of N with `algo_2` gets a performance of 382 GFLOP/s and the complete simulation sustained 335 GFLOP/s (the peak performance is 672 GFLOP/s). The parallel application is now described, the description and performances of the algorithm were given and the high-scalability was shown. In the next section we are going to exhibit one example of the use of the simulator to better understand the bio-mathematical model.

5 Biological results

Validation

It is the process of comparing the outputs of the model with the behavior of the phenomenon; in other words: comparing model execution to reality (*e.g.* field data). For example, we have observed in our simulations some behaviors of the output variables (ambiguous oscillations) which do not correspond to any observed biological phenomena. The parallel simulator allows us to track the origin of these results and to understand better the mathematical model.

Let us refer to hosts which have more than 300 parasites as "infected". Their strength is actually affected by parasites that aggregate on them. From biological experimentations, we know that the ratio of larvae recruited by the more parasitized hosts is greater than the ratio of larvae recruited by other hosts. We can fix the ratio $F\_old(x)$ of larvae recruited by hosts having more than 300 parasites, as a function of x, which is the ratio of hosts having more than 300 parasites. The shape of $F\_old(x)$ was fixed in the model and the only input parameter was the inflexion point of the curve presented on Fig. 20.

In Fig. 20, the inflexion point $(x_{if}, F\_old(x_{if}))$ has been set at (0.2, 0.8). This means that when 20% of the hosts have more than 300 parasites, then these hosts recruit 80% of the set of larvae present in the race-way. So the more larvae hosts have, the more they will recruit. A typical situation consists in a population within which only a few hosts are highly parisitized. Such an aggregation of parasites happens in natural conditions and in fish farms. The developpment of the parasite population depends on its reproduction success on hosts. Then, interactions between hosts and parasites at indivual scale influence dramatically the dynamics of parasite infrapopulations on these hosts, and finally the population dynamics at the host population scale.
The function F_old is used to precompute at each time step the function $\varphi(l, p, t)$ which is the probability for a host with p parasites to recruit l larvae; this function φ is used in the update of N. Figures 21 and 22 show two output variables of the same simulation, which corresponds to a given set of parameters where $(x_{if}, F\_old(x_{if})) = (0.20, 0.95)$. We will give some interpretation of these curves and figure out which is the best for our problem.

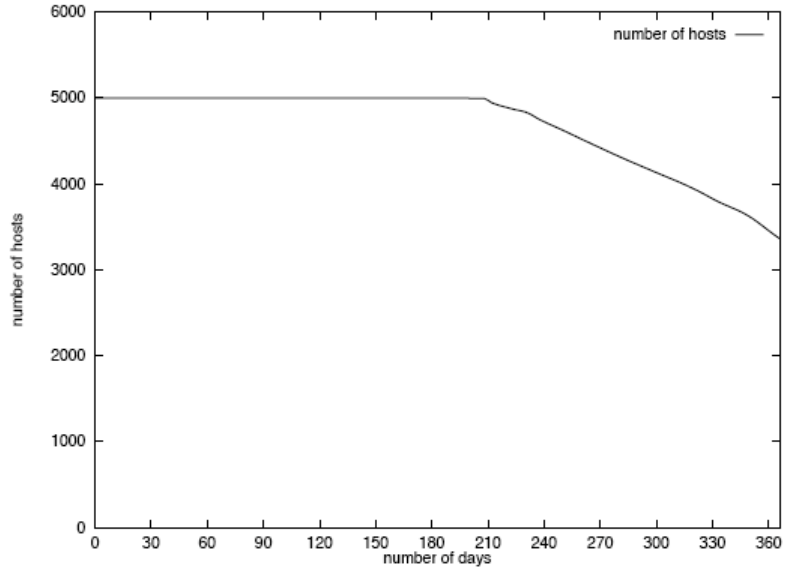

Figure 21: Output variable : number of hosts for one simulation

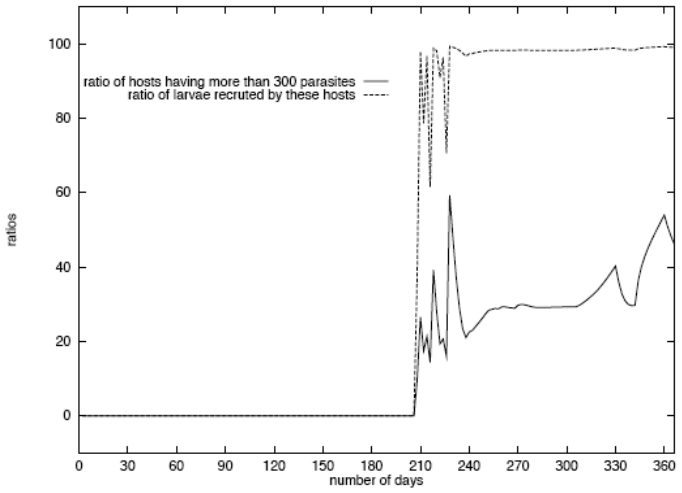

Figure 22: Output variable: ratio x of hosts having more than 300 parasites and function $F\_old(x)$

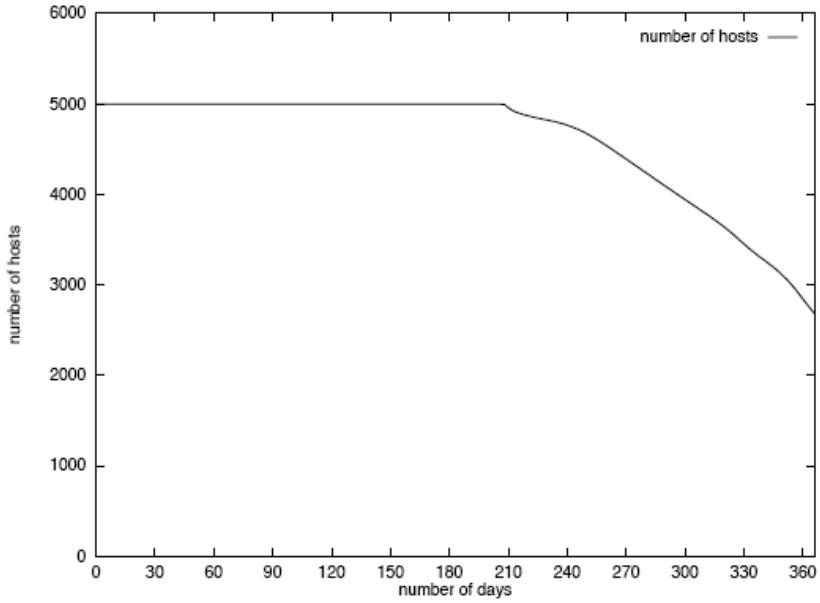

Figure 23: Output variable : number of hosts for one simulation

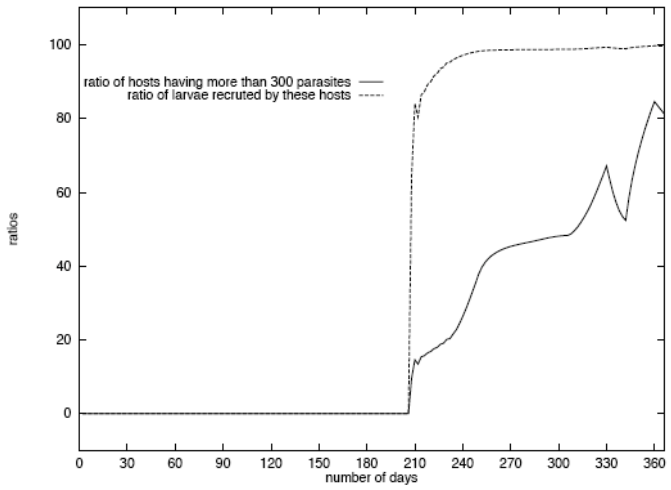

Figure 24: Output variable : ratio x of hosts having more than 300 parasites, and function $F\_new(x)$

During time steps from 200 to 250 days, we observe that variable x exhibits sudden starts on one time step when x increases towards x_{1x}. This is strange and could be a numerical artifact. Let us imagine the following scenario, the variable x increases and get closer and closer to x_{1x}. For an increasing Δx in the variable x, the function $F\_old(x)$ increases approximately by $\Delta F\_old = (F\_old)'(x)\,\Delta x$. If the derivative of F_old has a high value for $0 < x < x_{1x}$ as it can be seen from Fig. 20, and for x close to x_{1x}, one gets a relative big positive value of ΔF_old. We can analyze the consequences of this feature. This large value ΔF_old implies a higher aggregation, which means if the ratio of infested hosts is slowly increased then these hosts get many more parasites than in the previous time step. Consequently there is a sudden death of hosts having more than 300 parasites and then the ratio x of this type of hosts decreases. Finally, if $x < x_{1x}$ is near x_{1x}, a growth of the variable x may imply a decrease of x at the next time step.

Let us imagine another scenario: the variable $x_{1x} < x$ decreases and gets closer and closer to x_{1x}. For a decrease Δx in the variable x, the function $F\_old(x)$ decreases approximately of $\Delta F\_old = (F\_old)'(x)\,\Delta x$. For x just below x_{1x}, one gets a relative large negative value for ΔF_old. The large decrease of $F\_old(x)$ implies a smaller aggregation, which means that larvae can be recruited by many hosts and not only the more infected ones. Then, some hosts get more than 300 parasites, and then the ratio x of this type of hosts increases. Last, if $x > x_{1x}$ is near x_{1x}, a decrease in the variable x may imply an increase of x at the next time step. So we have illustrated an instability at the point $x = x_{1x}$ and characterized a phenomena of repel. This numerical approach is not satisfying from a biological point of view because it seems to have no biological basis. The oscillations have a period of $2\,\Delta t$ implying a numerical artifact. So we need to choose another function F_new with a nearly similar shape to F_old to replace it. The function F_new is displayed on Fig. 20 and has a derivative equal to one for $x = x_{1x}$. With this new function, the simulation leading to the curves of Fig. 21 and 22 was recomputed without modifying the set of parameters and gave the ones of Fig. 23 and 24. The behaviour of ratio x is lifted and the ambiguous oscillations disappeared.

Other minor changes have been performed during the validation of the model; some of them could be found with a more biologically-oriented discussion in [13].

Calibration

It is the process of parameter estimation for a model. Calibration is a tuning of existing parameters and usually does not involve the introduction of new ones (this could change the model structure). The set of parameters we used has been fixed from average parameter estimation [5].

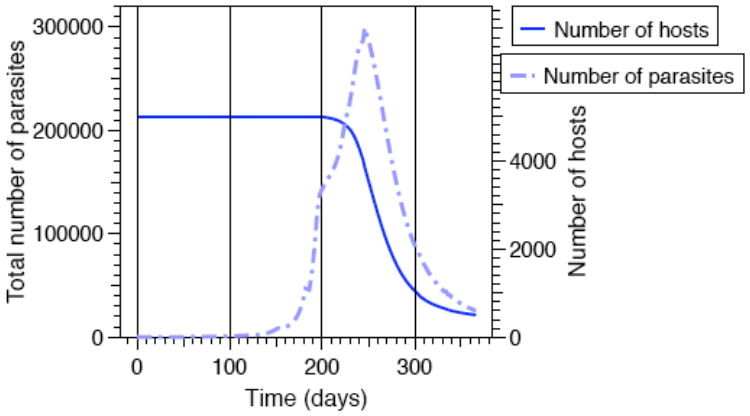

Figure 25: Temporary endemic state at the end of simulation

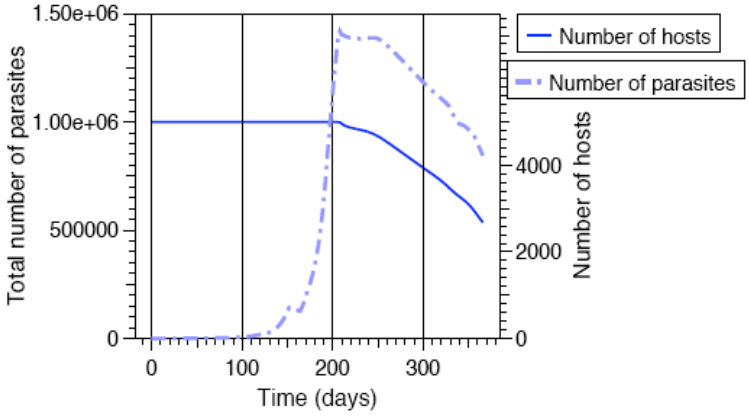

Figure 26: Parasite epidemic regulated by host deaths

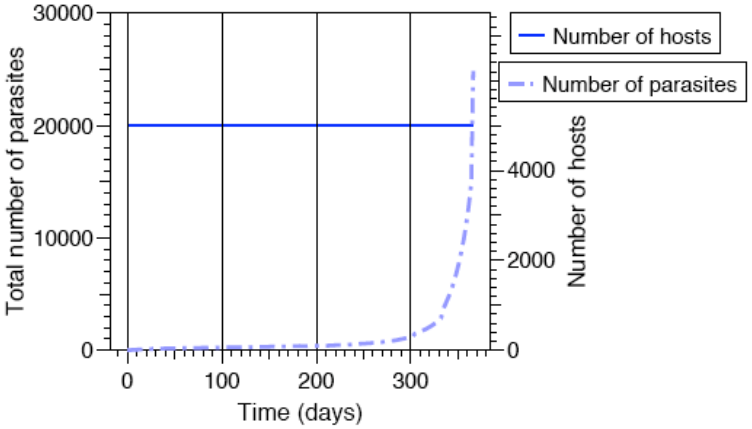

Figure 27: Growth of parasite population without epidemic

Results

The results consists in new dynamics actually observed with the new parallel simulator [23]. With the initial sequential simulator, only two dynamical behaviors were observed: first, host extinction because of a parasite epizootic process, second, parasite extinction without any host death. With the parallel version, the computations are more accurate (approximations are removed), and fast computations lead to the simulation of many realistic dynamics. Thus, we have obtained three other dynamic behaviors corresponding to observed biological cases:

- the fish population does not go extinct after introduction of parasites. A host population regulation is observed and a temporary endemic state is achieved (Fig. 25);
- deaths in the host population regulate the parasite epizootic process. Tuning the parameters controlling the aggregation of parasites allows us to obtain a more or less strong impact on the host population (Fig. 26);
- the parasite population develops, but it has no impact on the host population dynamics (Fig. 27).

The simulation validates some of the observed population dynamics and allows us to explore the model. Moreover, we have noticed through simulations that a variable aggregation has a central position in the hierarchy of the various mechanisms involved in the host-parasite system as biologists have expected.

6 Conclusion

A parallel numerical simulator based on a bio-mathematical model of a host-parasite system was presented. We studied a task mapping using a bidimensional snake distribution that leads theoretically to a very good load balancing for the main computation of the simulator. The performance analysis established the efficiency and high scalability of the parallel algorithm. A complete and costly simulation of 105 TFLOP lasts less than 9 minutes on the SGI Origin 3800 and 6 minutes 30 secondes on the IBM SP3 using 448 processors (double-precision reals). Furthermore, for simulations of 1.45 PetaFLOP, the simulator achieved a sustained performance of 50% of the peak on 448 processors.

So this parallelization work leads to a usable research tool and to a better understanding of how the bio-mathematical model reacts. The parallel simulator allows us to compute complex and non-linear interactions between different ecological levels. A feedback from the simulation results to the model allowed us to improve the assumptions initially taken. Numerical artifacts were then suppressed. Future work will include a study of the acceptable variability of the parameters, and a sensivity analysis with respect to initial conditions (that

is, inference about how simulation responds to changes for one or more inputs parameters). For this population dynamics problem, another simulation model [10] based on a Monte-Carlo algorithm with a different computation cost will be compared with the one presented here.

References

[1] S. Al-Battran, A.J. Field, R.L. Wiley, and J. Woods. Parallel simulation of plankton ecology. In *Proceedings of the IASTED International Conference Modelling And Simulation, Philadelphia, USA*, pages p. 259–263, May 1999.

[2] E. Anderson, Z. Bai, J. Bischof, J. Demmel, J.J. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK User's Guide*, SIAM Publications, 2nd edition, 1995.

[3] G. Booth. Gecko: A continuous 2-d world for ecological modeling. *Artificial Life Journal*, 3(3):147–163, 1997.

[4] C. Bouloux. *Modélisation, simulations et analyse mathématique de systèmes hôtes-parasites.* PhD thesis, Université Bordeaux I, December 1997.

[5] C. Bouloux, M. Langlais, and P. Silan. A marine host-parasite model with direct biological cycle and age structure. *Ecological Modelling*, 107:73–86, 1998.

[6] R. Costanza and T. Maxwell. Distributed modular spatial ecosystem modelling. *International Journal of Computer Simulation. Special Issue on Advanced Simulation Methodologies*, 5(3):247–262, 1995. *http://iee.umces.edu/SME3/.*

[7] R.M. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):31–53, October 1990.

[8] A. Grama, A. Gupta, V. Kumar, and G. Karypis. *Introduction to Parallel Computing: Design and Analysis of Algorithms, Second Edition.* Addison Wesley, 2003.

[9] M. Langlais, G. Latu, J. Roman, and P. Silan. Parallel numerical simulation of a marine host-parasite system. In *Europar'99 Parallel Processing*, volume 1685 of *Lecture Notes in Computer Science*, pages 677–685. LNCS 1685 - Springer Verlag, 1999.

[10] M. Langlais, G. Latu, J. Roman, and P. Silan. Performance analysis and qualitative results of an efficient parallel stochastic simulator for a marine host-parasite system. *Concurrency & Computation: Practice and Experience*, 15(11-12):1133–1150, september 2003.

[11] M. Langlais and P. Silan. Theoretical and mathematical approach of some regulation mechanisms in a marine host-parasite system. *Journal of Biological Systems*, 3(2):559–568, 1995.

[12] G. Latu. Solution parallèle pour un problème de dynamique de population. *Technique et Science Informatiques*, 19:767–790, June 2000.

[13] G. Latu. *Algorithmique parallèle et calcul haute performance dédiés à la simulation d'un système hôte-macroparasite*. PhD thesis, Université Bordeaux 1, December 2002. *http://icps.u-strasbg.fr/~latu/Thesis*.

[14] Lawrence Lau. *Implementation of Scientific Applications on Heterogeneous Parallel Architectures*. PhD thesis, Department of Mathematics, University of Queensland, Australia, November 1996. *http://www.acmc.uq.edu.au/~ll/thesis/*.

[15] S. A. Levin, editor. *Mathematics and Biology: The Interface - Challenges and Opportunities*. Lawrence Berkeley Lab PUB-701, 1992. *http://www.bis.med.jhmi. edu/Dan/mathbio/intro.html*.

[16] H. Lorek and M. Sonnenschein. Using parallel computers to simulate individual-oriented models in ecology: a case study. In *Proceedings: ESM'95 European Simulation Multiconference, Prag*, June 1995.

[17] B. Maniatty, B. Szymanski, and T. Caraco. Tempest: A fast spatially explicit model of epidemics on parallel machines. In A. Tentner, editor, *Proc. High Performance Computing Symposium, San Diego*, pages 114–119. SCS, 1994.

[18] Message Passing Interface Forum. MPI: A Message-Passing Interface Standard. version 1.1 June 12, 1995. Available by anonymous ftp from *ftp.mcs.anl.gov*.

[19] Message Passing Interface Forum. MPI: A Message-Passing Interface Standard. *International Journal of Supercomputer Applications*, 8(3-4), 1994. Also available by anonymous ftp from *ftp.netlib.org*.

[20] N. Minar, R. Burkhart, C. Langton, and M. Askenazi. The swarm simulation system: A toolkit for building multi-agent simulations. Technical report, Santa Fe Institute, Santa Fe, June 1996. *http://www.santafe.edu/projects/swarm/*.

[21] J.D. Mulder, J.J. Van Wijk, and R. Van Liere. A survey of computational steering environnment. *Future Generation Computer Systems*, 15(1):119–129, 1999.

[22] D.M. Nicol and R.M. Fujimoto. Parallel simulation today. *Annals of Operations Research*, 53:249–286, December 1994. *http://www.cc.gatech.edu/computing/pads/papers.html*.

[23] P. Silan, H. Caltran, and G. Latu. *Taxonomy, ecology and evolution of metazoan parasites*, chapter - Ecologie et dynamique des populations de Monogènes, ectoparasites de Téléostéens marins : approche et contribution montpelliérainne. Presses Universitaires de Perpignan, 2003. C. Combes & J. Jourdane editors.